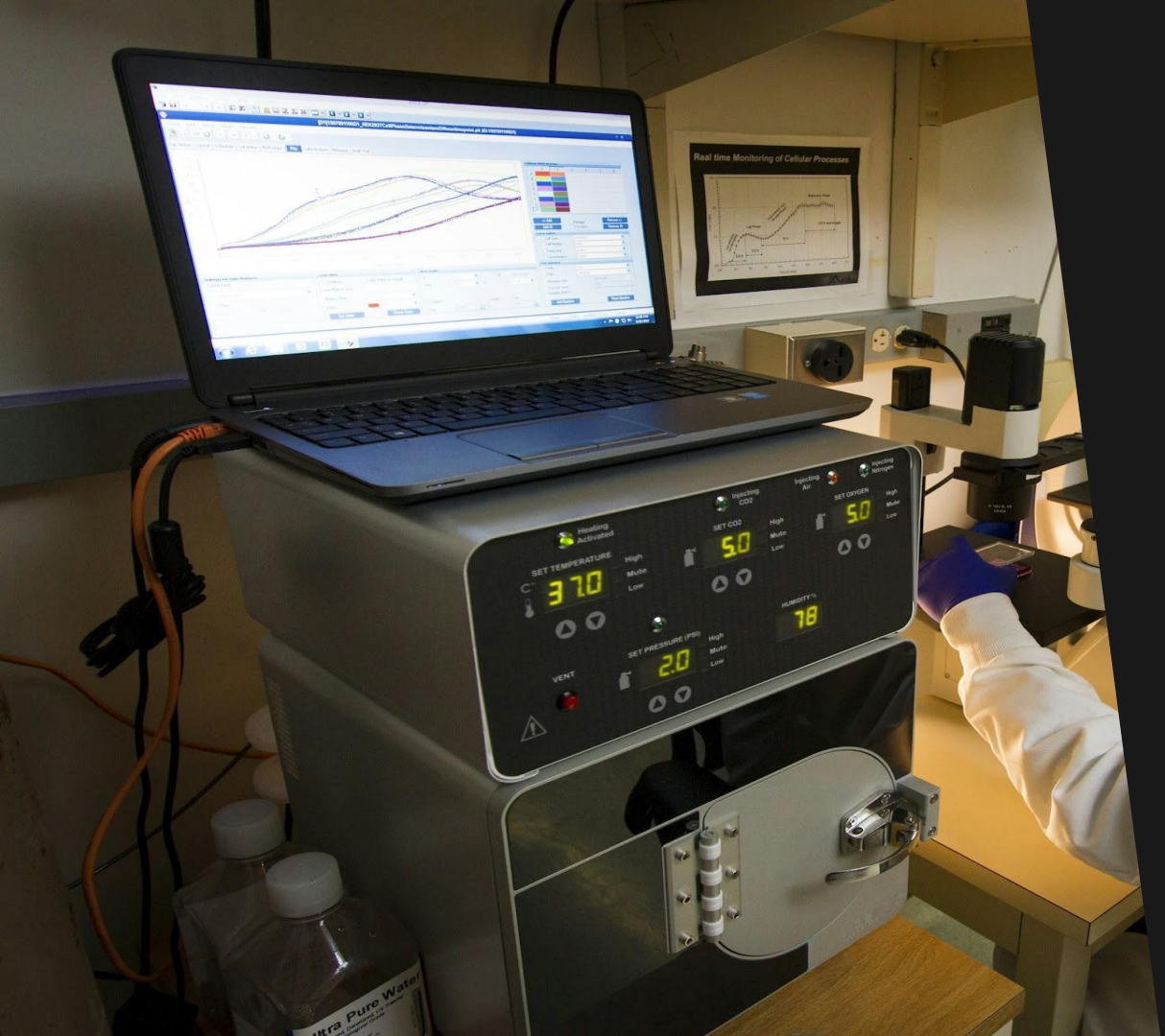
Maîtrise des Grands Projets Informatiques

Plan du cours

- Introduction aux Concepts de Base de l'Organisation des Entreprises
- Cycle de vie et cycle de développement logiciel
- Activités de développement logiciel
- Avant-projet
- Collecte des besoins : cas d'utilisation, user's story
- **Analyse Conception**
- Codage
- Intégration
- Déploiement et production : plateforme de déploiement continu, surveillance des applications, conteneurisation en production (docker)
- Activités de gestion de projets


Real time Monitoring of Cellular Processes
SET TEMPERATURE
37.0
SET CO2
5.0
SET OXYGEN
5.0
SET PRESSURE (PSI)
2.0
HUMIDITY %
78
VENT

Analyse

Analyse Conception

Elle consiste à comprendre les besoins et les exigences des utilisateurs finaux et à les traduire en spécifications détaillées.

- L'analyse permet de s'assurer que le produit final répondra aux attentes des utilisateurs.
- Elle aide à éviter les erreurs coûteuses et les retards dans le développement.

Rappel des différentes étapes de l’analyse

- Collecte des Besoins
- Analyse des Besoins
- Validation des Besoins
- Documentation des Exigences

Rappel des outils pour la collecte des besoins

- Interviews : Discussions structurées avec les utilisateurs pour comprendre leurs besoins.
- Questionnaires : Enquêtes écrites pour recueillir des informations sur les besoins.
- Ateliers : Sessions de travail collaboratives pour explorer les besoins en groupe.
- Observation Directe : Observation des utilisateurs dans leur environnement de travail pour comprendre leurs besoins.

Classification et priorisation

Une fois les besoins récoltés, il faudra les classifier et les prioriser dans le projet.

Cette priorisation peut s’appuyer sur des critères techniques comme fonctionnels.

Classification des besoins

Les besoins doivent être classifiés en différentes catégories :

- Besoins fonctionnels : ce que le système doit faire.
- Besoins non fonctionnels : comment le système doit le faire.
- Besoins techniques : contraintes techniques liées à l'infrastructure, à la compatibilité, ou à l'intégration avec d'autres systèmes.

Outils d’analyse des besoins

- La matrice des besoins
- La méthode Moscow

Matrice des besoins

La matrice d’analyse des besoins est un outil utilisé pour structurer et prioriser les besoins d’un projet. Elle permet d’organiser les informations recueillies lors de l’identification des besoins et d’évaluer leur importance et leur faisabilité.

Colonnes de la matrice des besoins

- Besoins / Exigences
- Priorité
- Critères de succès
- Responsable
- État
- Notes / Justifications

Etapes d’utilisation de la matrice des besoins

- Étape 1 : Recueil des besoins
- Étape 2 : Remplissage de la matrice
- Étape 3 : Validation
- Étape 4 : Mise à jour

Exemple de matrice des besoins

Besoin	Priorité	Critère de succès	Responsable	Etat	Notes
Authentification des utilisateurs	Haute	Tous les utilisateurs peuvent se connecter	Equipe 1	En cours	Basé sur l'authentification unique
Interface utilisateur	Moyenne	Temps de chargement < 3 s	Equipe 2	Non Commencé	Retour utilisateurs nécessaires
Sécurité des données	Haute	Aucune fuite des données	Equipe 3	En cours	Révision par l'équipe 2

Méthode MoSCow

La méthode MoSCoW est une technique de priorisation utilisée pour classer les exigences et les besoins d’un projet. Elle aide les équipes à déterminer quelles fonctionnalités doivent être développées en premier, en fonction de leur importance et de leur impact.

Catégories du MoSCow

- Must have (Doit avoir)
- Should have (Devrait avoir)
- Could have (Pourrait avoir)
- Won't have (N’aura pas)

Etapes du MoSCow

- Étape 1 : Recueil des besoins
- Étape 2 : Catégorisation
- Étape 3 : Validation
- Étape 4 : Mise à jour

Exemple de MoSCow

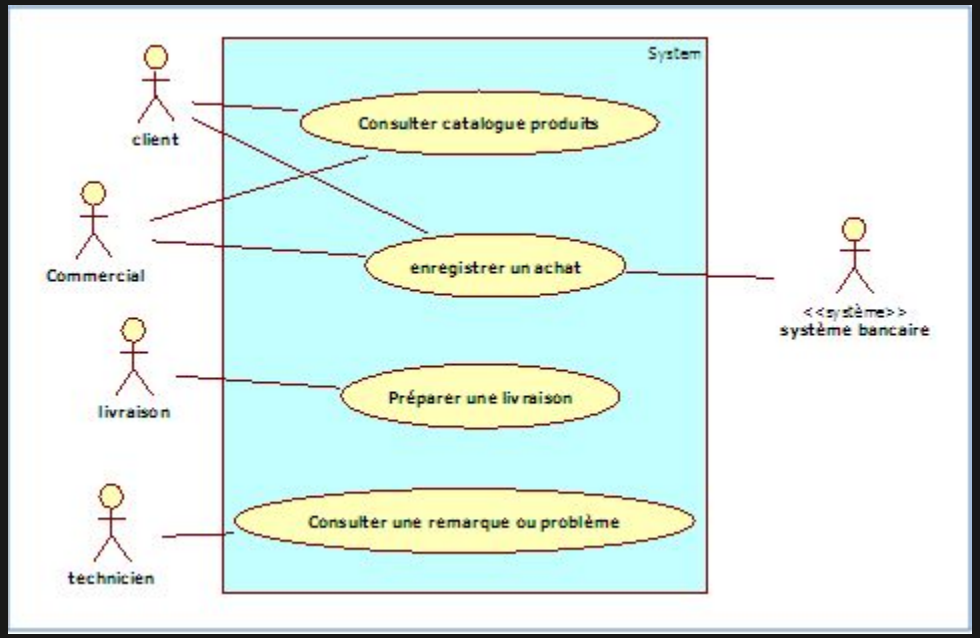
- **Must Have:** Les utilisateurs doivent créer de nouvelles tâches
- **Should Have:** Envoi de rappels aux utilisateurs pour les tâches à venir
- **Must Have:** Marquer des tâches comme complètes
- **Could Have:** Intégration avec des calendriers externes
- **Won’t Have:** Support de l'espéranto dans la version initiale.

Rappels de la modélisation des besoins

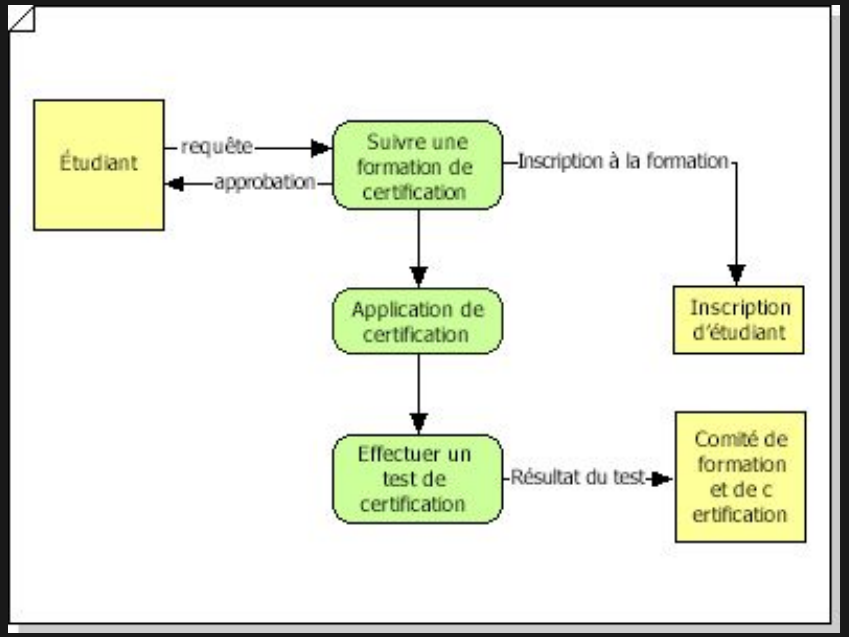
La modélisation des besoins est une étape essentielle pour assurer une compréhension claire des exigences d'un système.

Celle-ci peut s’utiliser à l’aide d’outils tels que les diagrammes UML.

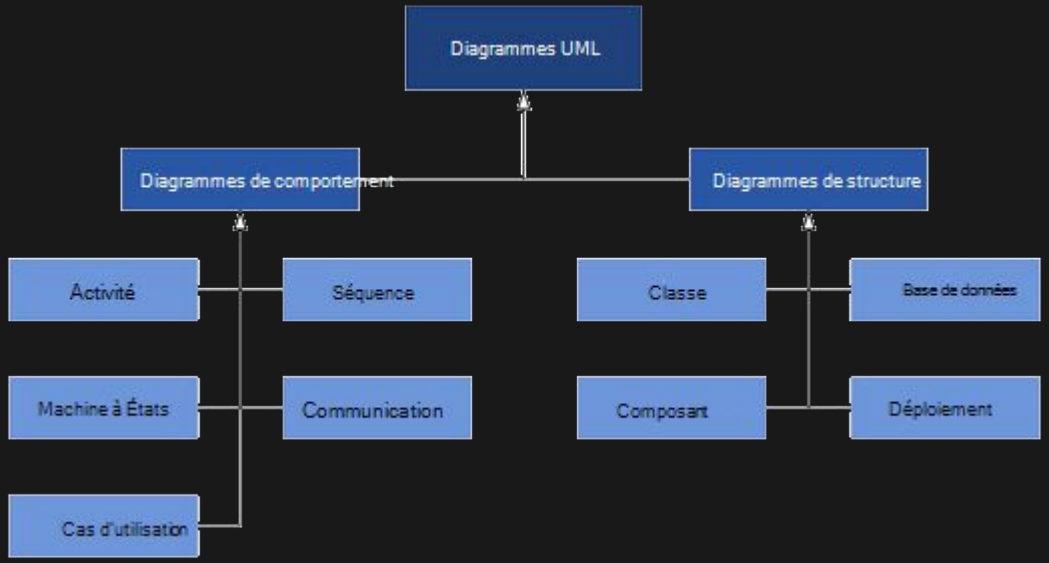
Rappel diagramme de cas d'utilisation



Rappel diagramme de flux de données



Rappel diagrammes UML



Validation des besoins

Elle vise à garantir que les exigences exprimées sont claires, complètes, cohérentes et réalisables. Cela permet de s'assurer que le système ou produit développé correspondra aux attentes des utilisateurs et des parties prenantes.

Une mauvaise validation des besoins peut entraîner des retards, des dépassements de coûts ou des fonctionnalités non conformes.

Objectifs de la validation des besoins

- Complètes : Toutes les attentes des parties prenantes ont été exprimées et prises en compte.
- Cohérentes : Les exigences ne se contredisent pas et sont logiquement reliées entre elles.
- Réalisables : Les besoins exprimés peuvent être mis en œuvre dans les contraintes techniques, temporelles et budgétaires du projet.

Objectifs implicites

- Confirmer que toutes les parties prenantes comprennent et valident les exigences.
- Identifier et corriger les lacunes, incohérences ou malentendus dans les exigences.
- Assurer que les besoins non fonctionnels (comme la sécurité, la performance ou la maintenabilité) sont bien pris en compte.

Méthodes de validation des besoins

- Revue des exigences avec les parties prenantes
- Prototypage

Revue des exigences

La revue des exigences est une étape collaborative durant laquelle les parties prenantes (clients, utilisateurs, développeurs, chefs de projet, etc.) examinent ensemble la liste des exigences pour valider leur adéquation au projet.

Méthodes de revue des exigences

- Organiser des réunions avec toutes les parties prenantes.
- Passer en revue chaque exigence de manière détaillée, clarifier les points flous.
- Prioriser les exigences pour mieux comprendre les fonctionnalités critiques par rapport aux fonctionnalités optionnelles.

Livrables de revue des exigences

- Document révisé des exigences (souvent un cahier des charges fonctionnel ou un document de spécifications).
- Compte-rendu de réunion avec les corrections, ajouts ou suppressions apportés.

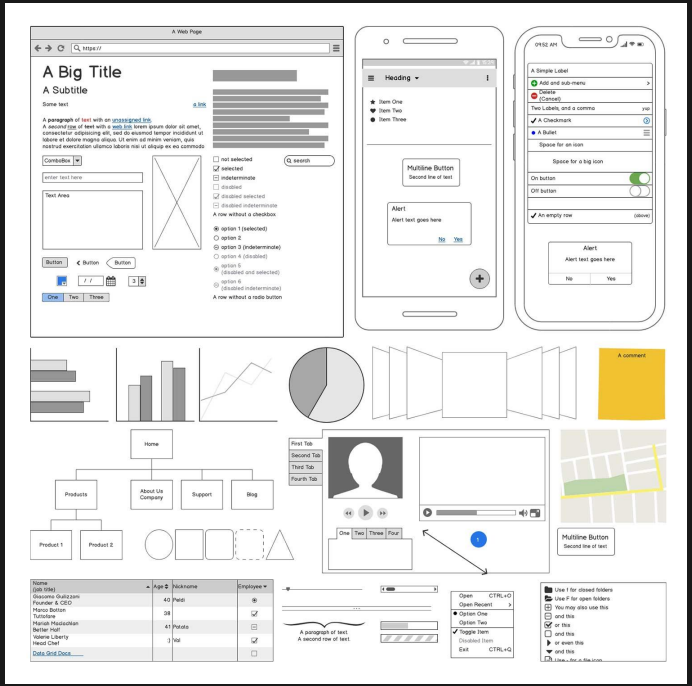
Prototypage

Le prototypage est une méthode puissante pour la validation des besoins, car il permet aux utilisateurs de visualiser et d’interagir avec une représentation simplifiée du produit final.

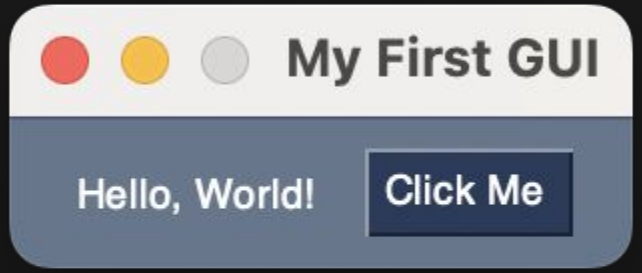
Types de prototypages

- Prototypage rapide : Création de maquettes ou de wireframes pour illustrer les interfaces ou les interactions clés.
- Prototypes interactifs : Simuler des interactions basiques (ex. navigation entre pages, interaction avec les menus) pour que les utilisateurs puissent tester et donner leur avis sur l'expérience utilisateur.

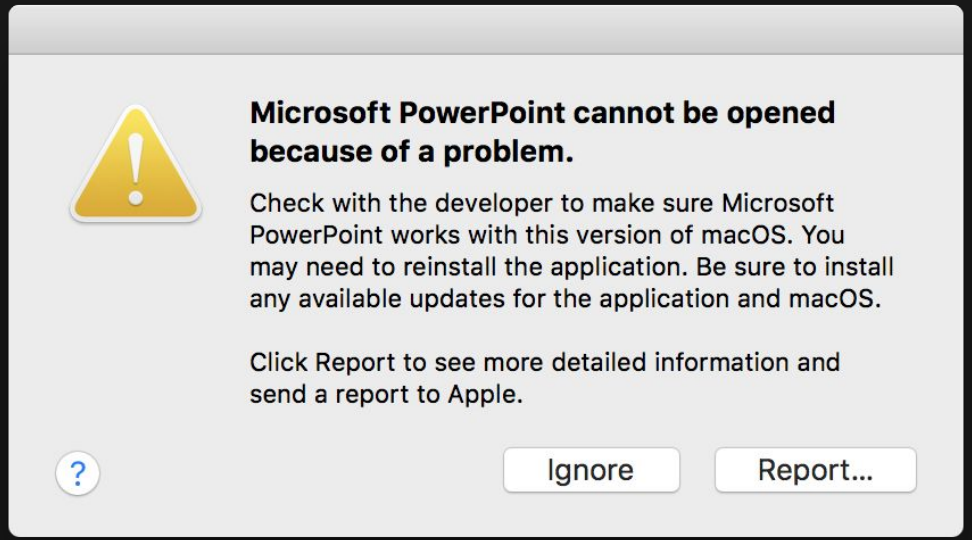
Exemple de prototypage rapide



Exemple de prototypage interactif



Exemple de prototypage interactif



Conception

Définition de Conception

La conception et l'architecture des systèmes jouent un rôle crucial dans le développement de logiciels modernes. Elles garantissent que les solutions technologiques proposées sont robustes, évolutives, et maintenables.

Les frameworks, quant à eux, offrent des outils pour accélérer le développement, en fournissant des structures prédéfinies et des composants réutilisables.

Conception

La conception consiste à traduire les besoins fonctionnels en une architecture technique qui définit comment les différents composants du système interagissent entre eux.

Cela inclut les algorithmes, les structures de données, les interfaces utilisateur, les bases de données, et plus encore.

Principes clés de la conception logicielle

- **Modularité** : Découper le système en composants ou modules distincts qui interagissent entre eux mais restent indépendants pour faciliter la maintenance et l'évolutivité.
- Cohésion et couplage :
  - **Cohésion** : Chaque module doit avoir une responsabilité bien définie.
  - **Couplage** : Les interactions entre modules doivent être minimisées pour éviter une forte interdépendance.

Principes clés de la conception logicielle

- **Réutilisabilité** : Conception de composants pouvant être réutilisés dans plusieurs parties du projet ou dans d'autres projets.
- **Abstraction** : Dissimuler les détails complexes de l'implémentation pour offrir une interface simple aux autres modules.
- **Encapsulation** : Limiter l'accès direct aux données d'un module, permettant seulement certaines interactions définies.

Modèles de conception

Les design patterns (modèles de conception) sont des solutions réutilisables pour résoudre des problèmes courants dans la conception logicielle.

Design patterns courants: Singleton

Problème : Comment s'assurer qu'une classe n'a qu'une seule instance ?

Solution : Le Singleton garantit qu'il n'existe qu'une seule instance d'une classe, accessible globalement.

Singleton: Exemple

```
class Singleton:
    _instance = None

    @staticmethod
    def get_instance():
        if Singleton._instance is None:
            Singleton._instance = Singleton()
        return Singleton._instance
```

Design patterns courants: Factory

Problème : Comment créer des objets sans spécifier directement leur classe ?

Solution : Utilisation de méthodes pour créer des objets, tout en laissant aux sous-classes la responsabilité d’instancier le bon type d’objet.

Factory: Exemple

```
class AnimalFactory:
    def create_animal(self, animal_type):
        if animal_type == "chien":
            return Chien()
        elif animal_type == "chat":
            return Chat()
```

Design patterns courants:
Observer

Problème : Comment s'assurer que plusieurs objets sont informés d'un changement d'état dans un autre objet ?

Solution : Le modèle Observer permet à un objet d’informer automatiquement plusieurs autres objets de ses changements d’état.

Observer: Exemple

```
class Sujet:
  def __init__(self):
    self.observateurs = []

  def ajouter_observateur(self, obs):
    self.observateurs.append(obs)

  def notifier(self):
    for obs in self.observateurs:
      obs.mise_a_jour()

class Observateur:
  def mise_a_jour(self):
    print("Observateur notifié")
```

Architecture des systèmes

L’architecture des systèmes concerne la structure globale du système, définissant les composants majeurs, leurs interactions, et la manière dont le système est organisé pour répondre aux exigences fonctionnelles et non fonctionnelles (sécurité, performance, scalabilité).

Le choix d’une architecture

Le choix d’une architecture est dicté à la fois par les exigences fonctionnels et les exigences non fonctionnelles.

Cela implique que la même architecture ne peut pas forcément être appliquée à une application mobile ou une application de gestion de logistique.

Exemples d’architectures courantes

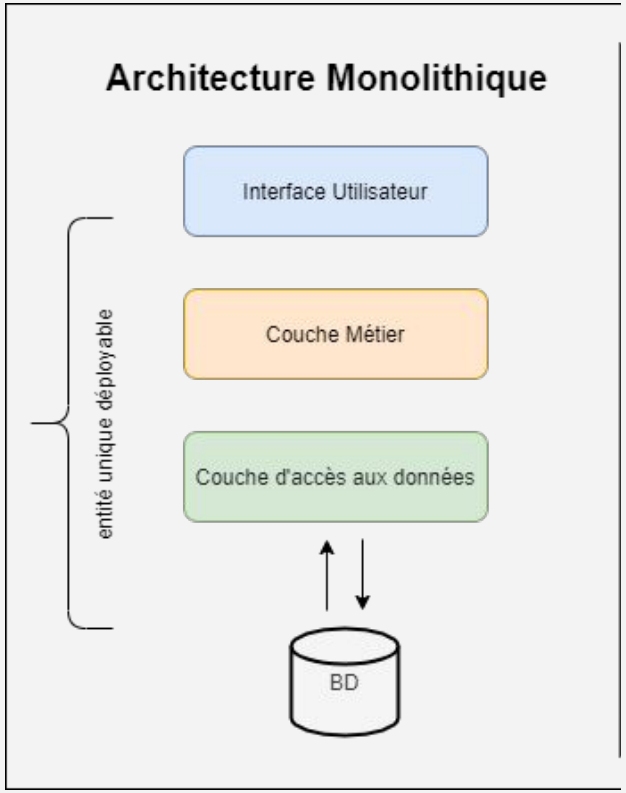
- Architecture monolithique
- Architecture orientée services (SOA)
- Architecture microservices
- Architecture événementielle

Architecture Monolithique

L’ensemble de l’application est construit comme un bloc unique où toutes les fonctionnalités sont regroupées.

Cette architecture est plus simple à développer mais présente les inconvénients d’être compliqué à maintenir et à faire évoluer.

Exemple d’Architecture Monolithique



Architecture orientée services (SOA)

Le système est décomposé en services indépendants qui communiquent via des protocoles (comme SOAP ou REST).

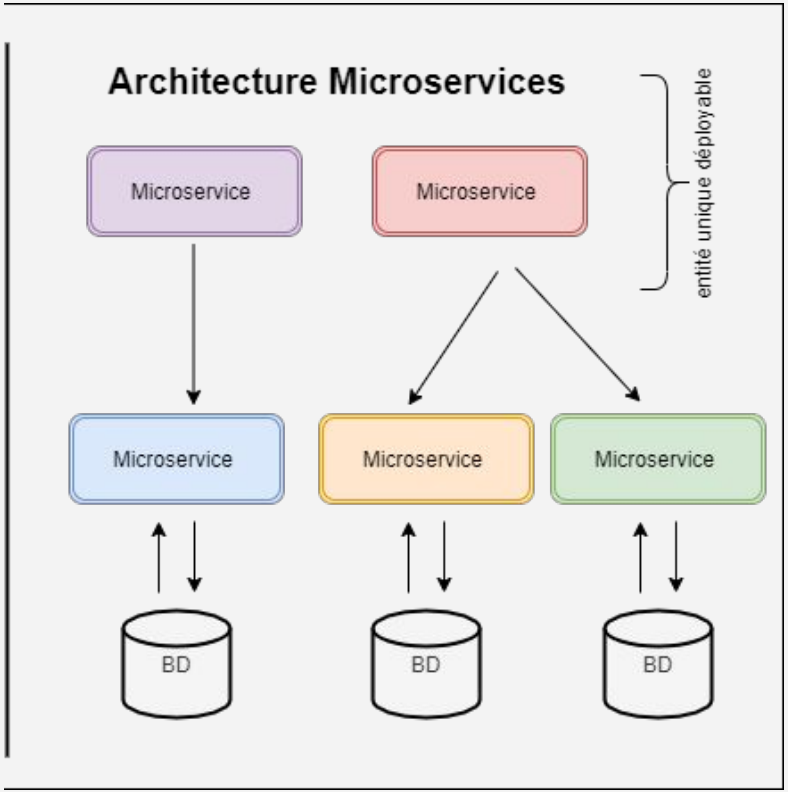
Cette architecture permet la modularité et la réutilisation des services mais introduit de la complexité dans la communication entre les services.

Architecture Micro service

Une forme d’architecture SOA où chaque service est déployé et maintenu indépendamment des autres.

Présente l’avantage d’être facile à faire évoluer au détriment de la complexité de mise en place et débogage

Exemple d'Architecture Microservice



Architecture événementielle

Utilise des événements pour communiquer entre les composants. Les événements représentent des changements d'état ou des actions à prendre.

Permet d’augmenter encore la décentralisation mais complexifie la gestion de la cohérence des données.